tensorimage Documentation

Release stable

Feb 16, 2019

Contents

CHAPTER 1

Configuration

There are some configurations that must be adjusted for certain TensorImage features to work in your computer. These can be modified with a script like the following:

```
import tensorimage as ti

# Update configurations
ti.config.manager.set_config(workspace_dir='/path/to/workspace/', tensorimage_path='/
↪path/to/tensorimage/')
# workspace_dir is the path to your workspace directory
# tensorimage_path is the path where TensorImage is stored

# Create workspace directory
ti.util.make_workspace.make_workspace()
```

Your workspace directory is simply a folder where all files, trained models and data relating to TensorImage are stored.

CHAPTER 2

Adding a training image dataset

All training datasets must have the following structure:

Example script for adding a training dataset to TensorImage:

```
import tensorimage as ti

dataset_path = '/home/user/Training datasets/MNIST/' # Path to training dataset with␣
↪images, should have structure as specified above
dataset_name = 'MNIST_training'
data_name = 'MNIST_training_data' # Unique name assigned to the specific set of data␣
↪that will be created by running this code once. It will be used later to specify␣
↪what data to use for training

ti.dataset.path_writer.write_training_dataset_paths(dataset_path, dataset_name)
ti.dataset.label_writer.write_labels(dataset_path, dataset_name)
image_loader = ti.image.loader.ImageLoader(data_name, dataset_name, 'training')
image_loader.extract_image_data()

image_writer = ti.image.writer.TrainingDataWriter(image_loader.image_data, data_name,␣
↪dataset_name, image_loader.img_dims)
image_writer.write_image_data()
```

CHAPTER 3

Adding an unclassified image dataset

All unclassified datasets must have the following structure:

Example script for adding an unclassified image dataset to TensorImage:

```
import tensorimage as ti

dataset_path = '/home/user/My unclassified datasets/MNIST/'
dataset_name = 'MNIST_unclassified'
data_name = 'MNIST_unclassified_data'

ti.dataset.path_writer.write_unclassified_dataset_paths(dataset_path, dataset_name)
image_loader = ti.image.loader.ImageLoader(data_name, dataset_name, 'unclassified')
image_loader.extract_image_data()
image_writer = ti.image.writer.DataWriter(image_loader.image_data, data_name, dataset_
↪name, image_loader.img_dims)
image_writer.write_image_data()
```

CHAPTER 4

Training

4.1 Without data augmentation

```
import tensorimage as ti

data_name = 'MNIST_training_data' # data_name assigned to extracted data previously
training_name = 'MNIST_train_op' # Unique name for 1 specific training operation that␣
↪will be used to identify trained models and other information for classification
n_epochs = 600
learning_rate = 0.04
l2_regularization_beta = 0.05 # Beta value for L2 Regularization (to prevent␣
↪overfitting)
architecture = 'RosNet' # Other CNN architectures are also available
batch_size = 32
train_test_split = 0.2

trainer = ti.train.trainer.Trainer(data_name=data_name, training_name=training_name,␣
↪n_epochs=n_epochs, learning_rate=learning_rate, l2_regularization_beta=l2_
↪regularization_beta, architecture=architecture, data_augmentation_builder=(None,␣
↪False), batch_size=batch_size, train_test_split=train_test_split, verbose=1)
trainer.build_dataset()
trainer.train()
trainer.store_model()
```

4.2 With data augmentation

```
import tensorimage as ti

data_name = 'MNIST_training_data' # data_name assigned to extracted data previously
training_name = 'MNIST_train_op' # Unique name for a specific training operation that␣
↪will be used to identify trained models and other information for classification
```

(continues on next page)

(continued from previous page)

```
n_epochs = 600
learning_rate = 0.04
l2_regularization_beta = 0.05 # Beta value for L2 Regularization (to prevent␣
↪overfitting)
architecture = 'rosnet' # Other CNN architectures are also available
batch_size = 32
train_test_split = 0.2
```

There are many data augmentation operations which you can perform on the training data. You can apply all of them to your training data, or just one, or none. You must pass the operation classes, with any required parameters, to the `DataAugmentationBuilder()` class, which will then be passed to the `Trainer()` class. The script continues below:

```
# Image flipping
image_flipper_op = ti.data_augmentation.ops.FlipImages()

# Pepper-salt noise
salt_vs_pepper = 0.1
amount = 0.0004
pepper_salt_noise_op = ti.data_augmentation.ops.AddPepperSaltNoise(salt_vs_
↪pepper=salt_vs_pepper, amount=amount)

# Random brightness
max_delta = 0.8
random_brightness_op = ti.data_augmentation.ops.RandomBrightness(max_delta)

# Gaussian blurring
sigma = 1
gaussian_blur_op = ti.data_augmentation.ops.GaussianBlur(sigma=sigma)

# More data augmentation operations are available, see documentation
data_augmentation_builder = ti.data_augmentation.builder.
↪DataAugmentationBuilder(image_flipper_op, pepper_salt_noise_op, lighting_
↪modification_op, gaussian_blur_op)

trainer = ti.train.trainer.Trainer(data_name=data_name, training_name=training_name,␣
↪n_epochs=n_epochs, learning_rate=learning_rate, l2_regularization_beta=l2_
↪regularization_beta, architecture=architecture, data_augmentation_builder=(data_
↪augmentation_builder, True), batch_size=batch_size, train_test_split=train_test_
↪split, verbose=1)
trainer.build_dataset()
trainer.train()
trainer.store_model()
```

The trained image classification model will be stored in the path:

```
workspace_dir/user/trained_models/training_name
```

4.3 Available architectures

The available architectures that can be passed to the `Trainer()` class `architecture` parameter are:

- RosNet ('rosnet')
- AlexNet ('alexnet')

CHAPTER 5

Training clusters

TensorImage allows you to quickly compare the performance of multiple trainers based on the testing accuracy, helping on hyperparameter optimization, as you will be able to know the hyperparameters the top trainers used. Here is an example script:

```
import tensorimage as ti

data_name = 'MNIST_training_data' # data_name assigned to extracted data previously, the same for all training_names

# Training operation 1 (without augmentation)
training_name_1 = 'MNIST_train_op_1' # training_name assigned to this specific training operation
n_epochs_1 = 600
learning_rate_1 = 0.05
l2_regularization_beta_1 = 0.04 # Beta value for L2 Regularization (to prevent overfitting)
architecture_1 = 'rosnet' # Other CNN architectures are also available

batch_size_1 = 32
train_test_split_1 = 0.2

trainer1 = ti.train.trainer.Trainer(data_name=data_name, training_name=training_name_1, n_epochs=n_epochs_1, learning_rate=learning_rate_1, l2_regularization_beta=l2_regularization_beta_1, architecture=architecture_1, data_augmentation_builder=(None, False), batch_size=batch_size_1, train_test_split=train_test_split_1, verbose=1)

# Training operation 2 (with data augmentation)
training_name_2 = 'MNIST_train_op_2'
n_epochs_2 = 1500
learning_rate_2 = 0.009
l2_regularization_beta_2 = 0.03 # Beta value for L2 Regularization (to prevent overfitting)
architecture_2 = 'RosNet' # Other CNN architectures are also available
batch_size_2 = 16
```

(continues on next page)

(continued from previous page)

```
train_test_split_2 = 0.3

# Building data augmentation operations
# Pepper-salt noise
salt_vs_pepper = 0.1
amount = 0.0004
pepper_salt_noise_op = ti.data_augmentation.ops.AddPepperSaltNoise(salt_vs_
↪pepper=salt_vs_pepper, amount=amount)

# Gaussian blurring
sigma = 1
gaussian_blur_op = ti.data_augmentation.ops.GaussianBlur(sigma=sigma)

data_augmentation_builder = ti.data_augmentation.builder.
↪DataAugmentationBuilder(pepper_salt_noise_op, gaussian_blur_op)

trainer2 = ti.train.trainer.Trainer(data_name=data_name, training_name=training_name_
↪2, n_epochs=n_epochs_2, learning_rate=learning_rate_2, l2_regularization_beta=l2_
↪regularization_beta_2, architecture=architecture_2, data_augmentation_builder=(data_
↪augmentation_builder, True), batch_size=batch_size_2, train_test_split=train_test_
↪split_2, verbose=1)

cluster_trainer = ti.train.cluster_trainer.ClusterTrainer(trainer1=trainer1,␣
↪trainer2=trainer2)
cluster_trainer.train()
results = cluster_trainer.get_results()
print(results)
```

Assuming that the training operation with data augmentation:

```
{
"1": {
   "name": "trainer2",
   "completed": True,
   "testing_accuracy": 0.97,  # Final testing accuracy
   "testing_cost": 38,  # Final testing cost
   "n_epochs": 1500,  # Epochs used by this trainer
   "learning_rate": 0.009,  # Learning rate used
   "l2_regularization_beta": 0.03,  # L2 Regularization beta value used
   "train_test_split": 0.3,  # Train-test split used
   "architecture": "RosNet",  # ConvNet architecture used
   "batch_size": 16  # Batch size used
   },
"2": {
   "name": "trainer1",
   "completed": True,
   "testing_accuracy": 0.87,  # Final testing accuracy
   "testing_cost": 832,  # Final testing cost
   "n_epochs": 600,  # Epochs used by this trainer
   "learning_rate": 0.05,  # Learning rate used
   "l2_regularization_beta": 0.04,  # L2 Regularization beta value used
   "train_test_split": 0.3,  # Train-test split used
   "architecture": "RosNet",  # ConvNet architecture used
   "batch_size": 32  # Batch size used
   }
}
```

CHAPTER 6

Classification

```
import tensorimage as ti

data_name = 'MNIST_unclassified_data' # data_name assigned to extracted data from␣
↪MNIST unclassified dataset
training_name = 'MNIST_train_op_2' # training_name assigned to training operation,␣
↪will be used to identify the trained model
classification_name = 'MNIST_classify_op' # Unique name assigned to this specific␣
↪classification operation
show_images = (True, 20) # Specifies if images with labels will be displayed, and the␣
↪maximum amount of random images to display

classifier = ti.classify.classifier.Classifier(data_name=data_name, training_
↪name=training_name, classification_name=classification_name, show_images=show_
↪images)
classifier.build_dataset()
classifier.predict()
classifier.write_predictions()
```

The final predictions for all of the unclassified images will be stored in the path:

```
workspace_dir/user/predictions/training_name/classification_name/
```

Depending on the size of your dataset, after several seconds an image with its predicted class as title should open.